
ACTIVE PATHS THROUGH MULTIMEDIA DOCUMENTS

POLLE T. ZELLWEGER

Xerox Palo Alto Research Center

ABSTRACT

We have developed a scripting mechanism for creating active paths through a document or set of documents. Scripted multimedia documents can contain a combination of text, graphics, audio, and actions. Scripts can be used in a wide variety of ways, such as for formal demonstrations and audio-visual presentations, for informal interpersonal communications, and for organizing collections of information. Scripted documents are a dynamic form of hypermedia document whose additional structure can be layered on top of existing documents.

1. Introduction

The advent of workstations has enabled a new and qualitatively different kind of document: a dynamic one that can incorporate audio and video in addition to text and graphics, one that can present itself in different ways depending on the needs or wishes of a particular reader at a particular time, and one that can serve as the backbone of a variety of "computations" that a reader might wish to perform. A prototype system for investigating these new capabilities has been built in the Cedar programming environment [Swinehart86] at the Xerox Palo Alto Research Center using the capabilities of the Etherphone voice system [Zellweger88] and the Tioga text editor [Teitelman84, Beach85].

1.1. *Overview of scripted documents*

A *script* is an active directed path through one or more documents that need not follow the linear order of the documents. Each entry in a script consists of a document location, such as a contiguous sequence of characters in a text document, together with an associated action and timing. Sample actions might play back a previously-recorded voice annotation, send text to a text-to-speech synthesizer, open a new window, animate a picture, or query a database. Script specifications are stored in a shared database separately from the underlying documents. A single document can have multiple scripts traversing it for different purposes.

A script can be played back as a whole, in which case the first document in the script is displayed on the screen, positioned to show the first location. The location is highlighted to call attention to it, and its associated action is performed. After the associated timing, the system highlights the location of the next script entry, scrolling if needed, performs its action, and so on. The same document location can appear at multiple points in the script, with the same or different associated actions and timing.

Arbitrary actions at a scripted location allow scripted documents to perform a wide variety of tasks, including demonstrations, tutorials, and programming tools. Parameterized actions allow a script to be personalized, such as "Hello <username>," or to more accurately reflect the current state of affairs, such as "There are <currentNumber> entries in this category." Scripts can be very formal items that are carefully crafted for pedagogical reasons, such as a videotape or a presentation, or they can be informal, used to communicate from a single script writer to a single script reader (who might well be the same person).

1.2. *Related work*

The capabilities of electronic documents have been expanding rapidly in recent years. Multimedia document systems have extended the contents of documents from text and formatting to include bitmap images, geometric graphics, spreadsheets, attributes, and voice [Ades86, Crowley87, Luther87]. Hypertext systems allow users to link non-contiguous portions of documents to express the associations between them [Delisle86, Halasz86, Meyrowitz86, Conklin87]. Electronic books and encyclopedias combine multimedia (text, graphic illustrations, and sometimes video) and hypertext (to link related sections) and may also include interactive portions in which readers can run simulations or other experiments to improve their understanding [Feiner82, Weyer85, Yankelovich85]. Presentation tools make it possible for users to capture sequences of actions to create automatic demonstrations [Xerox85]. Other document systems have included animation [Fiume87], actions [Hogg84], or sequencing [Christodoulakis86]. These advanced capabilities have been provided in several ways: by example, by direct manipulation, or by some form of programming language. The scripted documents system described in this paper forms a unique combination of multimedia documents, hypertext links, sequencing, and actions that increases document functionality still further.

2. Key ideas of scripted documents

This section discusses some of the key aspects of scripted documents, emphasizing ways in which scripted documents differ from typical hypertext systems.

2.1. *Directed paths versus browsing*

Most hypertext systems have concentrated on providing links between related nodes, creating an information space that a user can browse at will. The related concept of a *directed path* through a set of documents allows authors or script writers to include more structure. This structure consists of timed sequences of information, unrelated to the ordering of the underlying documents, to help the user understand the material. Although a few hypertext systems allow a sequence of links to be combined to form a path [Conklin87], we have instead made paths our primitive mechanism in order to explore the applications of sequentiality in electronic documents (see Section 3). For example, a later path entry can rely upon the user having seen earlier entries to overlay or animate previously-seen images or to abbreviate explanations.

There is a conceptual continuum from the directed mode of following paths to the browsing mode of exploring links. For example, although standard usage of scripted documents is to experience a path automatically, users can single-step through a group of directed paths to approximate the more conventional browsing paradigm. A more interesting combination of directed paths and browsing is an *interactive path*, in which the user answers questions to construct his or her desired path through the information. In fact, a hypertext link can be expressed as a degenerate script containing two locations that have no corresponding actions.

2.2. *Emphasis on voice*

As a result of the efforts of the Etherphone project, recorded and synthesized voice are widely available in the Cedar environment. Documents prepared with the Tioga editor can contain formatted text, graphics, and an unlimited number of arbitrary-length voice annotations. A direct-manipulation voice editor allows easy editing at the phrase or sentence level.

Voice is a critical component of a script. It provides a unifying thread for presentations and interpersonal communication, and it provides an out-of-band way to organize and comment on written material. Finally, it is easy to collect and modify (but as yet hard to search, although speech recognition systems are rapidly improving).

2.3. *Script information separate from underlying ordinary documents*

Separating scripts from the underlying documents enables an interesting mixture of public and private scripts. Scripts can refer to public documents without modifying those documents, allowing a user to create a personally organized information space (such as during an authoring task) without copying information into a closed hypertext system. Separating scripts from the underlying documents also allows for a smooth integration of scripted documents with other documents: scripts can gradually be added to a set of documents that continue to be accessed as ordinary documents.

2.4. *Arbitrary actions accompany document locations*

Script actions are unconstrained—they can call upon the power of the surrounding computing environment to execute commands or program fragments. The following section illustrates the resulting flexibility and power of scripts.

3. Examples of scripts

Scripts have a wide variety of uses. Script writers can create formal teaching materials and documentation. Users can build scripts that organize information or perform repetitive tasks. In addition, programs can construct scripts to ease complex tasks.

Teaching tool. A language dialog can be represented both textually and as a simultaneous sequence of voice annotations to demonstrate correct pronunciation. The same dialog can include multiple scripts, one for each language desired. Each script visits the same sequence of locations, but has different associated voice annotations.

Interpersonal communication. To review a manuscript, each reviewer can prepare a script for the manuscript, including voice annotations as well as branches through other supporting documents. Each script can follow an arbitrary path through the manuscript to collect related points. This use provides much of the value of a face-to-face interaction between the reviewer and the author, in which the reviewer makes comments while flipping back and forth through the manuscript and other documents to substantiate those comments.

Personal information management. A user can create multiple scripts through a set of documents, each organizing a different topic. These scripts can be reordered as needed. The use of voice to annotate each location can be particularly helpful in early stages of idea exploration.

Audio-visual presentation. Voice, text, and graphics can be combined to simulate a “slide show” on a selected subject. Several versions of the slide show, including different or rearranged slides, can be constructed to accommodate different audiences or different time constraints.

User documentation and demonstration. Scripts can be used to explain how to use a complex program, such as a graphic illustrator. Actions can load and start the program, apply it to an example, visit noteworthy places in the user manuals, and explain some beginner’s projects. Different scripts can be prepared for novice, intermediate, and expert users. Figure 1 shows the first few entries of such a script applied to the tutorial for the Gargoyle illustrator [Pier88].

“Bouncing ball” for songs or poems (looping actions). The text of a song or poem with a refrain can be successively highlighted with simultaneous audio, returning to the single copy of the text for the refrain as appropriate. Representing the song or poem in

this way emphasizes the identity of the refrain, so that the reader need not carefully compare repeated copies of a linear representation. Similarly, a presentation may repeatedly return to an overview slide to orient viewers or to emphasize important points.

Program-constructed scripts as history mechanisms. Programs can automatically construct scripts to act as history mechanisms for their users. For example, searching through annotations in a voice-annotated document can be tedious. An annotation system could automatically construct three scripts whenever voice annotations were added to a document. The first would connect all annotations to the document in document order. The second would connect all annotations to the document in the order that they were made. A third more ephemeral script would cross document boundaries to connect all annotations made during a single session in the order that they were made.

Scripts can also be used for collecting locations of interest rather than for their sequencing and/or action capabilities. Consider the following two examples.

Program debugging tool (repetitive actions). To provide an easy way of controlling a named group of breakpoints during program debugging, a collection of locations in several program files can be joined into a script with actions that set appropriate breakpoints: standard, conditional, profiling, tracing, etc. This script can be executed whenever a user wishes to activate those breakpoints. They can later be removed without disturbing other active breakpoints by overriding the script's stored actions with a separately-specified action that clears the breakpoint.

Program editing tool (empty actions). A compiler could build a script containing the locations of all the syntax errors in a group of files compiled together. The user could then single-step through the script, making corrections along the way. The script both makes it easy for the user to find the next error location and automatically manages the changing character positions as the user adds or removes characters to correct the errors.

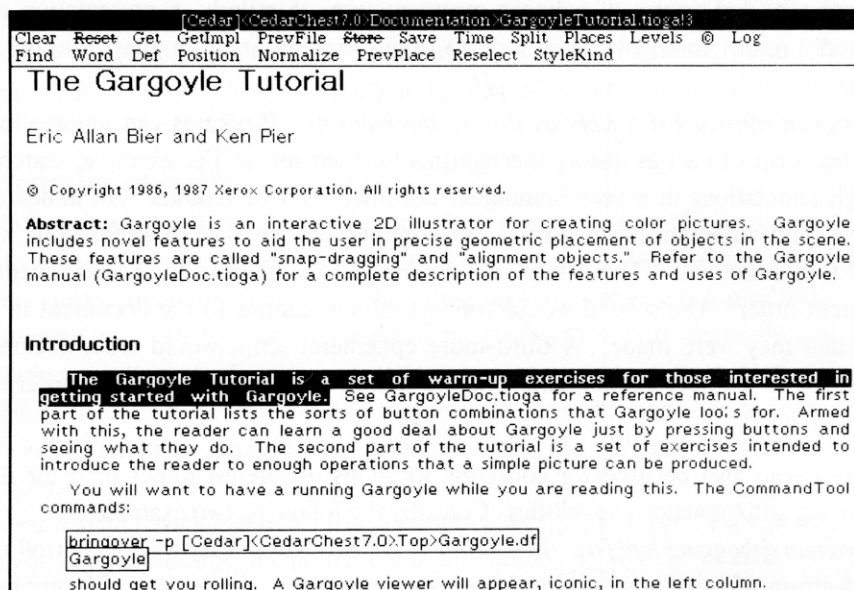
4. Creating and playing back scripts

4.1. Script creation and editing

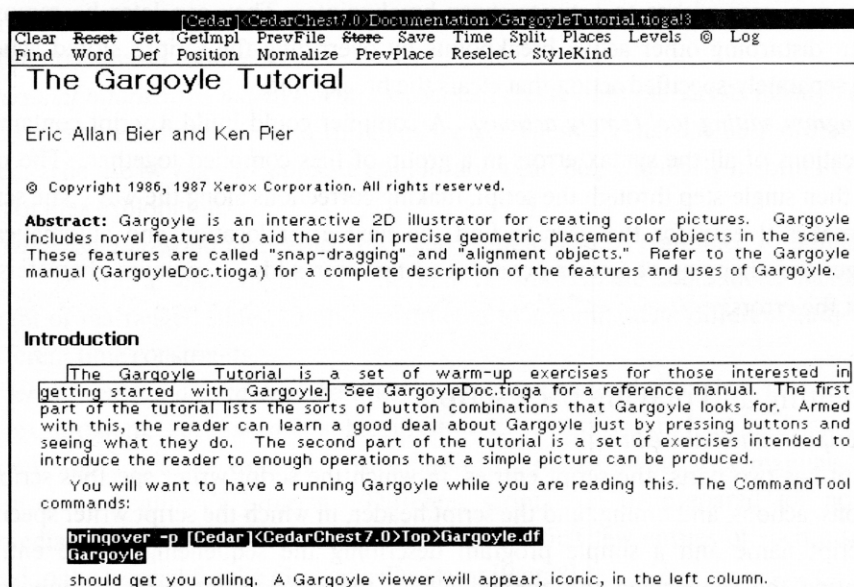
A script has two parts: the script entries, in which the script writer specifies scripted locations, actions, and timing; and the script header, in which the script writer specifies the script name and a simple program describing the sequencing of the entries. Separating the sequencing information allows the same script entry to appear in multiple scripts or multiple times in a single script.

A script tool is used to create and edit scripts. The prototype Script Tool is a form-based tool with a simple set of operations.

Figure 1a: A sample user documentation script, part 1.

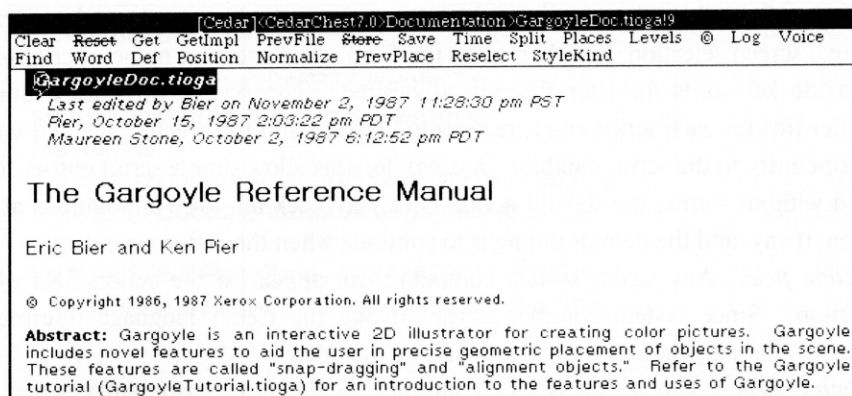


Sample script at the first entry. This action uses the text-to-speech synthesizer to greet the user by name, welcoming him or her to the Gargoyles tutorial.

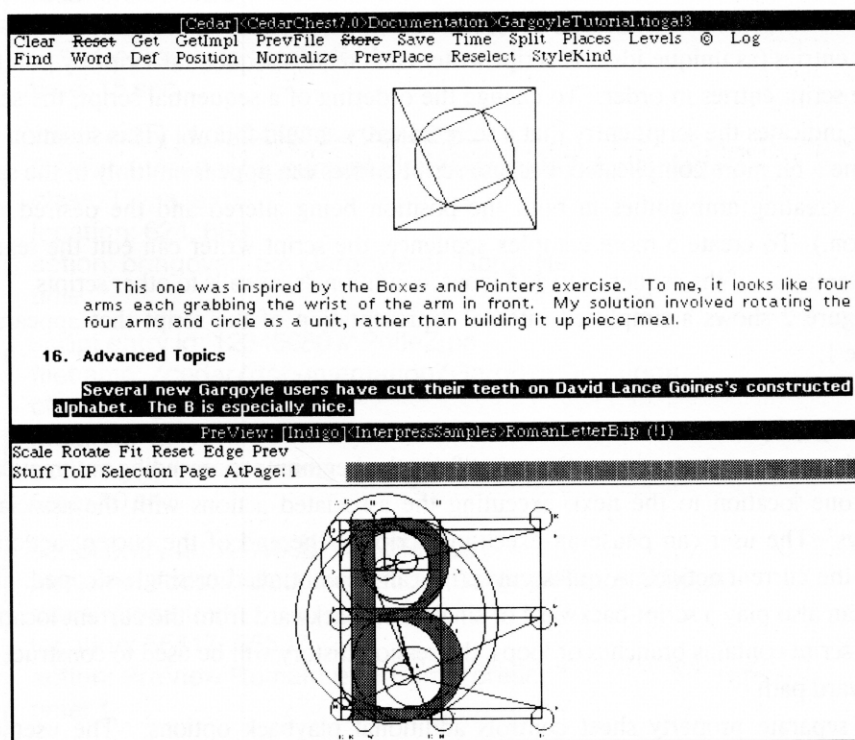


Sample script at the second entry. This action retrieves the executable files of the Gargoyles illustrator and starts the program (the same actions that the text is instructing the user to perform).

Figure 1b: A sample user documentation script, part 2.



Sample script at the third entry. This is a different file. The action plays back the previously-recorded voice annotation on the first character of the scripted location (a tiny "word balloon" around the character G indicates the presence of voice). The voice annotation explains the difference between the Gargoyles tutorial and the Gargoyles reference manual.



Sample script at the fourth entry. The script has returned to the tutorial document. The action displays on the screen an Interpress master for an image that was not included in the tutorial itself.

Script entries. To create or edit a script entry, the script writer specifies its action and timing fields by filling in the fields of a script entry form, sets its location by making a screen selection, and then saves the form. The user can name a script entry or provide keywords for later filtering, if desired. The system provides a unique identifier (id) for each script entry, records the creator and the create time, and writes the script entry to the script database. System defaults allow simple script entries to be created without forms: the default action is to play back all voice annotations at the location, if any, and the default timing is to continue when the action completes.

Action field. Any Cedar system command can appear in the action field of an annotation. Since system commands can invoke the Cedar language interpreter, arbitrary Cedar language statements can also be included.

Timing field. The script system continues to the next script entry when the specified duration has passed, regardless of whether the action has completed. The script writer can specify "*" to continue whenever the action completes or "~" to wait for user confirmation before continuing.

Script sequencing. The sequencing information for a script is specified separately from the script's entries. For a strictly sequential script, it consists of an ordered list of script entries (as unique ids). A script writer can create a sequential script by pointing to the script entries in order. To change the ordering of a sequential script, the script writer indicates the script entry that a selected entry should follow. (This situation can become a bit more complicated, because script entries can appear multiply in the same script, creating ambiguities in both the position being altered and the desired new position.) To create a more complex sequence, the script writer can edit the textual representation of the sequence to add loops, conditionals, or calls to other scripts.

Figure 2 shows a simplified internal representation of the script that appears in Figure 1.

4.2. Script playback

The Play command plays an entire script from the beginning, proceeding automatically from one location to the next, executing the associated actions with the associated timings. The user can pause an executing script at the end of the current action or abort the current action. A quiescent script can be continued or single-stepped. The user can also play a script backward or single-step backward from the current location. If the script contains branches or loops, the session history will be used to construct the backward path.

A separate property sheet controls additional playback options. The user can inhibit the actions associated with annotations (so as simply to traverse through the scripted locations), specify a single action to be executed at every scripted location, and increase or decrease the timing by some factor.

Figure 2: *Simplified internal representation of the script for Figure 1.**Script header*

```
script header id: 12345677 # PolleZ.pa
script name: Introduction to Gargoyle
file names:
  /cedar/documentation/GargoyleTutorial.tioga
  /cedar/documentation/GargoyleDoc.tioga
script sequence:
  12345678 # PolleZ.pa  12345679 # PolleZ.pa
  12345680 # PolleZ.pa  12345681 # PolleZ.pa
```

Script entries

```
script entry id: 12345678 # PolleZ.pa
filename: /cedar/documentation/GargoyleTutorial.tioga
class: Tioga text
location: 15..121
action: speak("Hello, ", UserCredentials.Get[].name,
  ". Get ready to blast off.")
time: 15 sec

script entry id: 12345679 # PolleZ.pa
filename: /cedar/documentation/GargoyleTutorial.tioga
class: Tioga text
location: 624..683
action: bringover -pm Gargoyle.df; Gargoyle
time: *

script entry id: 12345680 # PolleZ.pa
filename: /cedar/documentation/GargoyleDoc.tioga
class: Tioga text
location: 1..17
action: play(annotations)
time: 10 sec

script entry id: 12345681 # PolleZ.pa
filename: /cedar/documentation/GargoyleTutorial.tioga
class: Tioga text
location: 55417..55537
action: PreView RomanLetterB.interpress
time: *
```

4.3. *Script visualization and navigation*

Script visualization is a problem for both the script reader and the script writer, because script sequencing and script actions are not directly visible through examination of the document. The script reader must be able to tell that a document has associated scripts and which scripts are appropriate for him or her to play back. The script writer has the more difficult problem, in that he or she must also have tools for debugging faulty scripts.

The script reader is alerted to the presence of scripts in a document in two ways. First, when a document is displayed, a **Script** button appears in the document header if and only if the document has scripted locations. All script examination and playback operations are available from a popup menu generated by clicking the **Script** button. Second, each scripted location is distinctively marked. Textual scripted locations are marked with a surrounding rectangle; other scripted objects may be marked differently. The marker indicates the presence of one or more script entries at that location, which may appear at multiple places in multiple scripts. The user can view all script entries that include that scripted location or list the names of all scripts that include it. The reader can also ask what scripts have entries in a given document.

In addition, a user can browse the script database for script names, script entry names, keywords, or any other script field. Other navigation commands show the user the document location associated with a script entry, all scripts the entry belongs to, and all possible preceding or following entries in a given script.

5. Implementation

Scripted documents are implemented in the Cedar programming environment in the Computer Science Laboratory of the Xerox Palo Alto Research Center [Swinehart86]. The two systems that they rely upon most heavily are the Tioga editor and the Etherphone system. We describe each of these systems briefly, and then we describe the implementation of scripted documents.

5.1. *The Tioga editor*

The Tioga editor is a WYSIWYG "what you see is what you get" galley editor used for both program text and high-quality documents [Teitelman84, Beach85]. Tioga documents are tree-structured to express the organization of the document into sections, subsections, paragraphs, and so on. Tioga documents can be displayed at any level of detail, omitting nodes that are nested more deeply than the selected level.

Tioga documents can contain rich formatting and typography. Each node has an associated format, which specifies such parameters as its leading, margins, default typeface, and so on. A document as a whole has an associated style, which defines the

meanings of all formats used in that document. Nodes can also have arbitrary named properties, added by the user or by programs. These properties are not directly visible when the document is viewed, but a separate Edit Tool can be used to examine their values. One use of node properties is to specify images, Interpress masters, and additional parameters to support illustrations embedded in Tioga documents.

Individual characters in a Tioga document can have looks, which specify special typeface parameters such as boldface, italic, subscript, and the like. Characters can also have arbitrary named properties. Tioga has search commands that allow rapid searching of documents for given node or character properties.

5.2. *The Etherphone system*

The experimental Etherphone system uses Ethernet communications to transmit digitized voice [Zellweger88]. The system consists of microprocessor-based electronic telephones, a centralized switching server, a voice file server, and workstation programs to support voice communications and voice recording services. From a workstation, a user can place and receive telephone calls, maintain private telephone directories, and manage a database of voice messages. A voice annotation package allows voice to be added to Tioga documents and provides a simple direct-manipulation interface for editing voice [Ades86]. Furthermore, a commercial text-to-speech synthesizer exists as a server in the Etherphone network. The synthesizer allows the system to "speak" text, initiated either by the user (perhaps by selecting the text in a viewer) or by a program (such as speaking an error message or proofreading a document).

This work on scripted documents began as a project to exploit the capabilities of the Etherphone system by creating narrated documents. Narrated documents are scripted Tioga documents with a single type of action: playing back previously-recorded voice annotations.

5.3. *Scripted documents*

The two parts of a script specification, the script entries and the script sequencing information, are stored separately from the underlying documents in a simple B-tree-indexed database [Terry88]. The database's ability to merge the results of queries to multiple databases allows users to simultaneously access private databases containing their private scripts and public databases containing public scripts.

5.3.1. *Tagged and absolute references to scripted locations*

An important goal of the scripting system was to allow script writers to create script entries that refer to any document they can access, including documents they cannot or do not wish to modify (hereafter called *read-only* documents). This capability is especially important when scripts are being used as an organizational tool, such as

during an authoring task. Ideally, the owners of a scripted document would also be able to edit unscripted portions of the document without damaging the script.

Given our additional desire to use the existing version-based file system in our widely-distributed environment, we achieve reasonable functionality by allowing script entries to contain two different kinds of references to scripted locations. *Absolute references* refer to precise locations in documents, such as character or byte positions, while *tagged references* refer to uniquely tagged and hence movable objects within documents.

The initial prototype of the scripting system used only tagged references and stored each document's script entries within the document. However, the desire to script read-only documents suggested database storage for their script entries, and it proved more convenient in the later implementation to store all script entries together.

Absolute references are used to refer to items within read-only documents. The scripting system records the scripted location and the exact timestamp and version of the containing file. The file is considered immutable: if it is subsequently updated, the script will continue to refer to the older version. Note that many public documents, such as user documentation and reports, are likely to change less frequently than personal documents. We are exploring ways to handle such documents more robustly, such as also recording a signature containing the scripted location to permit the location to be found in the new version if it exists.

By contrast, creating a tagged reference to a scripted location modifies its containing file. The scripting system writes the unique id of the script entry in the scripted object. Tagged references allow other portions of the file to be edited without disturbing the scripted location, even when the scripting system is not running. However, editing the scripted location itself generally destroys its pointer to the script entry. These semantics are reasonable, because it is not clear that the new version of the scripted location is related to the old script entry. For example, consider scripting the word "reindeer" in a document to have an action that plays a bit of "Rudolph the Red-nosed Reindeer," and suppose that later edits change "reindeer" to "moose."

For either kind of reference, if a scripted location cannot be found during playback, the system displays an informational message and continues with the next script entry.

5.3.2. Scripting different kinds of document content

The design of the scripting system is object-oriented to permit scripting a variety of documents, such as Tioga documents and VLSI diagrams, and a variety of contents, such as text, bitmaps, and synthetic graphics. Each class of visual object must implement the following functions:


```
identify [object] -- returns class and location, suitable for storing
add [object, unique id] -- tag object with unique id
remove [object, unique id] -- remove tag
showWithHighlight [filename, filedate and version, location, unique id]
    -- used to highlight the object when it is the script entry being executed
```

The `identify` function takes a selected object, such as a sequence of text characters or a piece of a bitmap, and returns both the class of that object and its location (a persistent way of addressing it), suitable for storing in a script. For a sequence of text characters, it returns a character position and a length, while for a piece of a bitmap, it returns the coordinates of its bounding box. This function supports absolute references to scripted objects and provides a hint and a printable value for the location of a tagged object.

The `add` function supports tagged references to scripted objects by writing into the specified object a unique id that identifies a script entry. For example, for text in a Tioga file, the `add` function uses Tioga's ability to associate arbitrary property-value pairs with any character. The `remove` function is the inverse operation.

A class's `showWithHighlight` function finds an object in a file using either a filedate and location (for an absolute reference) or a unique id (for a tagged reference). This function positions the file so that the corresponding object is visible and highlights the object.

5.3.3. Making scripted locations visible

To avoid unknowingly destroying scripted locations, script writers and other document editors must be able to see them. To make tagged references visible, the scripted object's `add` procedure is responsible for adding the visible scripted location indicator to the object. Similarly, the `remove` procedure removes the indicator when it removes the script entry id. Making absolute references visible presents more of a challenge. The script system must consult the database whenever a new remote file is opened to see if absolute references to this file exist. If so, it constructs a view of the scripted read-only document with script indicators added.

6. Status and future work

The initial prototype of the scripting system was designed for creating narrated documents. Scripts performed a single action, namely playing back voice annotations, at each scripted location. Each script was constrained to refer to a single document, although a document could contain multiple scripts. Script entries were stored in the document header and contained their own sequencing information; tagged references referred to scripted locations throughout the document. A simple script tool allowed

users to create, edit, play back, and navigate through scripts.

The current prototype separates sequencing information from the remainder of the script entry (to permit multiple scripts to share the same entry) and stores both in a database (to allow scripting read-only documents and to make it easier to refer to multiple documents in a single script). The system has been redesigned to permit scripting different classes of document content, but text is the only class that is currently handled. The language for describing complex sequencing is still in its infancy: the scripting system currently allows only sequential scripts, although they may visit the same location repeatedly. The implementation of the scripting system uses a variety of previously-existing Cedar packages: the Tioga text editor, the Etherphone telephone and voice management system, the LoganBerry database system, the Command Tool command interpreter (with its ability to execute Cedar language statements), and the FS version-based file system. Additional user interface features are still needed, as they have not progressed much from the earlier prototype. In particular, the more advanced user playback options, such as backward playback and variable playback timing control, have not yet been designed.

We are working on extending our prototype scripting tool to allow conditional and/or interactive scripts; better visualization of scripts for both script readers and writers, including browsing tools and visual displays of script sequencing; better control of screen and document layout at each script entry during script execution; and scripting other classes of documents, such as graphic illustrations or VLSI layouts.

7. Summary

The scripting concept unifies action, presentation, and hypermedia to form a useful and flexible mechanism for improving documents of the future. This novel mechanism allows writers to communicate additional information to readers. Scripted multimedia documents can contain any combination of text, graphics, audio, and action. Scripts need not follow the normal linear order of their associated documents. In addition, script writers can construct multiple viewing paths through documents for different readers and for different purposes.

The scripting mechanism can be widely applied to create electronic documents with increased capabilities. Scripted documents can orchestrate formal presentations, arrange informal communications, organize collections of information, and ease repetitive or complex tasks.

Acknowledgments

Dan Swinehart and Stephen Ades provided the initial impetus to create narrated documents. Jock Mackinlay suggested several improvements to the design of the later

scripted documents system. While I would like to thank the many contributors to the Cedar programming environment as a group, I would also like to single out a few whose work has been particularly critical for the scripting system: Rick Beach and Michael Plass provided consultation on its interaction with Tioga, Doug Terry's LoganBerry database package supplied just the needed functionality, and Russ Atkinson's efforts on the Command Tool and Cedar interpreter were invaluable. Pavel Curtis, Subhana Menis, and Ken Pier helped to clarify the exposition of this paper.

References

- [1] Ades, S. & Swinehart, D. (1986). Voice annotation and editing in a workstation environment, *Proc. AVIOS'86 American Voice Input Output Society Conf.*, Arlington, VA, 13-28. Also available as Xerox PARC Technical Report CSL-86-3.
- [2] Beach, R. (1985). *Setting tables and illustrations with style*, Ph.D. thesis, U. of Waterloo, Canada. Also available as Xerox PARC Technical Report CSL-85-3.
- [3] Christodoulakis, S., Ho, F. & Theodoridou, M. (1986). The multimedia object presentation manager of MINOS: a symmetric approach, *Proc. ACM SIGMOD'86 Conf.*, Washington, DC, 295-310.
- [4] Conklin, J. (1987). Hypertext: an introduction and survey, *IEEE Computer*, **20**, 9, 17-41.
- [5] Crowley, T., Forsdick, H., Landau, M. & Travers, V. (1987). The Diamond multimedia editor, *Proc. USENIX Technical Conf.*, Phoenix, AZ, 1-18.
- [6] Delisle, N. & Schwartz, M. (1986). Neptune: a hypertext system for CAD applications, *Proc. ACM SIGMOD'86 Conf.*, Washington, DC, 132-143.
- [7] Feiner, S., Nagy, S. & van Dam, A. (1982). An experimental system for creating and presenting interactive graphical documents, *ACM Trans. Graphics*, **1**, 1, 59-77.
- [8] Fiume, E. & Tsichritzis, D. (1987). Multimedia objects, *IEEE Office Knowledge Engineering Newsletter*, **1**, 1, 60-64.
- [9] Halasz, F., Moran, T. & Trigg, R. (1987). NoteCards in a nutshell, *Proc. ACM CHI + GI'87 Human Factors in Computing Systems and Graphics Interface Conf.*, Toronto, Canada, 45-52.
- [10] Hogg, J. & Gamvroulas, S. (1984). An active mail system, *Proc. ACM SIGMOD'84 Conf.*, Boston, MA, 215-222; also *SIGMOD Record*, **14**, 2.
- [11] Luther, W., Woelk, D. & Carter, M. (1987). MUSE: multimedia user sensory environment, *IEEE Office Knowledge Engineering Newsletter*, **1**, 1, 49-59.
- [12] Meyrowitz, N. (1986). Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework, *Proc. OOPSLA'86 Object-Oriented Programming Systems, Languages and Applications Conf.*, Portland, OR, 186-201; also *SIGPLAN Notices*, **21**, 11.
- [13] Meyrowitz, N. & van Dam, A. (1982). Interactive editing systems: part 1, *Computing Surveys*, **14**, 3, 321-352.
- [14] Pier, K., Bier, E. & Stone, M. (1988). Gargoyle: an interactive illustration tool, *Proc. EP'88 Int'l Conf. on Electronic Publishing, Document Manipulation, and Typography*, Nice, France.
- [15] Swinehart, D., Zellweger, P., Beach, R. & Hagmann, R. (1986). A structural view of the

-
- Cedar programming environment, *ACM Trans. Programming Languages and Systems*, **8**, 4, 419-490.
- [16] Teitelman, W. (1984). A tour through Cedar, *IEEE Software*, **1**, 2, 44-73.
- [17] Terry, D. & Swinehart, D. (1988). Managing stored voice in the Etherphone system, to appear in *ACM Trans. Computer Systems*, **6**, 1. An extended abstract appears in *Proc. of Eleventh ACM Symposium on Operating System Principles*, Austin, TX, 1987, 103-104.
- [18] Weyer, S. & Borning, A. (1985). A prototype electronic encyclopedia, *ACM Trans. Office Information Systems*, **3**, 1, 63-88.
- [19] Xerox Corporation. (1985). *ViewPoint 1.0 Demo Maker tool*, Xerox Corporation, P.O. Box 470065, Dallas, TX 75427.
- [20] Yankelovich, N., Meyrowitz, N. & van Dam, A. (1985). Reading and writing the electronic book, *IEEE Computer*, **18**, 8, 15-30.
- [21] Zellweger, P., Terry, D. & Swinehart, D. (1988). An overview of the Etherphone system and its applications, *Proc. 2nd IEEE Conf. on Computer Workstations*, Santa Clara, CA.